* 1. Find the second largest number.

#include <stdio.h>

int main() {

intarr[]={12,33,34,67,7,89,56},i,j,temp,n;

clrscr();

n=sizeof(arr)/sizeof(arr[0]);

for(i=0;i<7;i++){

for(j=1;j<=n-i-1;j++){

if(arr[j-1]>arr[j]){

temp=arr[j-1];

arr[j-1]=arr[j];

arr[j =temp;

}

}

}

printf("Second Largest is : %d",arr[n-2]);

getch()

return 0;

}

* 1. Reverse an array.

#include <stdio.h>

void arrrev(intarr[],intrevarr[],int n){

inti,j;

j=0;

for(i=n-1;i>=0;i--){

revarr[j]=arr[i];

j++;

}

}

int main() {

int arr[]={12,33,34,67,7,89,56},i,n;

clrscr();

n=sizeof(arr)/sizeof(arr[0]);

intrevarr[n];

arrrev(arr,revarr,n);

for(i=0;i<n;i++){

printf("%d ",revarr[i]);

}

getch();

return 0;

}

* 1. Create aNxN array V, in such a way that Vij = i-j and then find sum of the array element.

#include<stdio.h>

void main()

{

intn,sum=0;

clrscr();

printf("Enter n :");

scanf("%d",&n);

int a[n][n];

for(inti=0;i<n;i++){

for(int j=0;j<n;j++){

a[i][j]=i-j;

sum+=i-j;

}

}

printf("\nSum of the array is : %d",sum);

getch();

}

* 1. Find the length of string.

#include<stdio.h>

int strlen(char n[]){

int c=0;

while (n[c]!='\0'){

c++;

}

return c;

}

void main()

{

char name[20];

clrscr();

printf("Enter a name:");

gets(name);

printf("length of your name is: %d",strlen(name));

getch();

}

* 1. Check the string is palindrome or not.

#include<stdio.h>

#include<string.h>

int isPalindrome(char n[],int len){

int j=len-1,i;

int flag=1;

for(i=0;i<len/2;i++){

if(n[i]!=n[j]){

flag=0;

break;

}

j--;

}

return flag;

}

void main()

{

char name[20];

clrscr();

printf("Enter a name:");

gets(name);

printf("String Palindrome status is: %d",isPalindrome(name,strlen(name)));

getch();

}

* 1. Convert string into upper and lower case.

#include<stdio.h>

#include<string.h>

void uppLow(char name[]){

int i,ichar;

char upper[50],lower[50];

for(i=0;i<strlen(name);i++){

ichar=(int)name[i];

if(ichar>64 && ichar<91){

upper[i]=name[i];

lower[i]=(char)ichar+32;

}

else if(ichar>96 && ichar<123){

lower[i]=name[i];

upper[i]=(char)ichar-32;

}

else{

lower[i]=upper[i]=name[i];

}

}

upper[i]=lower[i]='\0';

printf("Name in Uppercase: %s",upper);

printf("Name in Lowercase: %s",lower);

}

void main()

{

char name[20];

clrscr();

printf("Enter a name:");

gets(name);

uppLow(name);

getch();

}

* 1. Copy one string into another.

#include<stdio.h>

void strcpy(char des[],char source[]){

int i=0;

while(source[i]!='\0'){

des[i]=source[i];

i++;

}

des[i]='\0';

}

void main()

{

char name[20],copyName[20];

clrscr();

printf("Enter a name:");

gets(name);

strcpy(copyName,name);

printf("Copied name: %s",copyName);

getch();

}

* 1. Program to implement linear search.

#include <stdio.h>

int i;

void linearSearch(int arr[]){

int temp,flag=1;

printf("Enter item to search:\n");

scanf("%d",&temp);

for(i=0;i<10;i++){

if(arr[i]==temp){

printf("\nfound at index: %d", i);

flag=0;

}

}

if(flag){

printf("\nNot found!");

}

getch();

}

int main(){

int arr[10];

clrscr();

for(i=0;i<10;i++){

printf("Enter arr item:\n");

scanf("%d",&arr[i]);

}

linearSearch(arr);

/\* while (i<10 && arr[i]!=temp){

i++;

}

if(i<10){

printf("found at index: %d",i);

}

else{

printf("Not found!");

} \*/

return 0;

}

* 1. Program to implement binary search.

#include <stdio.h>

int i;

void binarySearch(int arr[],int low,int high){

int mid,temp;

printf("Enter item to search:\n");

scanf("%d",&temp);

while(low<=high){

mid=(low+high)/2;

if(temp==arr[mid]){

printf("found at index %d",mid);

return ;

}

else if(temp<arr[mid])

high=mid-1;

else

low=mid+1;

}

printf("Not found!!!");

}

int main(){

int arr[10];

clrscr();

printf("Enter array item only in sorted manner!!!\n");

for(i=0;i<10;i++){

printf("Enter an item:\n");

scanf("%d",&arr[i]);

}

binarySearch(arr,0,9);

getch();

return 0;

}

* 1. Program to implement bubble sort.

#include <stdio.h>

void bubbleSort(int arr[]){

int i,j,temp;

for(i=0;i<10;i++){

for(j=0;j<9-i;j++){

if(arr[j]>arr[j+1]){

temp=arr[j];

arr[j]=arr[j+1];

arr[j+1]=temp;

}

}

}

}

void main(){

int i,arr[10];

clrscr();

for(i=0;i<10;i++){

printf("Enter array item: ");

scanf("%d", &arr[i]);

}

bubbleSort(arr);

printf("Sorted array is: ");

for(i=0;i<10;i++){

printf("%d ",arr[i])

}

getch();

}

* 1. Program to implement selection sort.

#include <stdio.h>

void selectionSort(int arr[]){

int i,j,min,temp;

for(i=0;i<9;i++){

min=i;

for(j=i+1;j<10;j++){

if(arr[j]<arr[min]){

min=j;

}

}

if(min!=i){

temp=arr[min];

arr[min]=arr[i];

arr[i]=temp;

}

}

}

void main(){

int i,arr[10];

clrscr();

for(i=0;i<10;i++){

printf("Enter array item: ");

scanf("%d",arr[i]);

}

selectionSort(arr);

printf("Sorted array is: ");

for(i=0;i<10;i++){

printf("%d ",arr[i]);

}

getch();

}

* 1. Program to implement insertion sort.

#include <stdio.h>

void insertionSort(int arr[]){

int i,j,key;

for(i=1;i<9;i++){

key=arr[i];

j=i-1;

while(j>=0 && arr[j]>key){

arr[j+1]=arr[j];

j--;

}

arr[j+1]=key;

}

}

void main(){

int i,arr[10];

clrscr();

for(i=0;i<10;i++){

printf("Enter array item: ");

scanf("%d",arr[i]);

}

insertionSort(arr);

printf("Sorted array is: ");

for(i=0;i<10;i++){

printf("%d ",arr[i]);

}

getch();

}

* 1. Program to implement merge sort.

#include <stdio.h>

void merge(int arr[],int beg,int mid,int end){

int l,r,i,temp[10];

for(l=beg,r=mid+1,i=beg; l<=mid && r<=end;i++){

if(arr[l]<=arr[r])

temp[i]=arr[l++];

else

temp[i]=arr[r++];

}

while(l<=mid)

temp[i++]=arr[l++];

while(r<=end)

temp[i++]=arr[r++];

for(i=beg;i<=end;i++)

arr[i]=temp[i];

}

void mergeSort(int arr[],int beg,int end){

int mid;

if(beg<end){

mid=(beg+end)/2;

mergeSort(arr,beg,mid);

mergeSort(arr,mid+1,end);

merge(arr,beg,mid,end);

}

}

void main(){

int i,arr[10],n;

clrscr();

for(i=0;i<10;i++){

printf("Enter array item: ");

scanf("%d", &arr[i]);

}

mergeSort(arr,0,9);

printf("Sorted array is: ");

for(i=0;i<10;i++){

printf("%d ",arr[i]);

}

getch();

}

* 1. Program to implement quick sort.

#include <stdio.h>

int partition(int arr[],int beg,int end){

int pivot = arr[beg];

int i= beg+1;

int j=end;

int temp;

do{

while(arr[i]<=pivot){

i++;

}

while(arr[j]>pivot){

j--;

}

if(i<j){

temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

}while(i<j);

temp=arr[beg];

arr[beg]=arr[j];

arr[j]=temp;

return j;

}

/\* int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high - 1; j++) {

if (arr[j] < pivot) {

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

\*/

/\* int partition(int arr[], int low, int high) {

int pivot = arr[low];

int i = (high + 1);

for (int j = high; j >= low + 1; j--) {

if (arr[j] > pivot) {

i--;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i - 1], &arr[low]);

return (i - 1);

}

\*/

void quickSort(int arr[],int beg,int end){

if(beg<end){

int p = partition(arr,beg,end);

quickSort(arr,beg,p-1);

quickSort(arr,p+1,end);

}

}

void main(){

int i,arr[10],n;

clrscr();

for(i=0;i<10;i++){

printf("Enter array item: ");

scanf("%d",&arr[i]);

}

quickSort(arr,0,9);

printf("Sorted array is: ");

for(i=0;i<10;i++){

printf("%d ",arr[i]);

}

getch();

}

1. Write a c program to implement Linked List.

#include <stdio.h>

#include <stdlib.h>

struct list{

int data;

struct list \*next;

}\*head;

void insInEmpty(int data){

head = (struct list \*)malloc(sizeof(struct list));

head->data=data;

head->next=NULL;

printf("Inserted successfully.\n");

}

void insAtBeg(int data){

if(isEmpty()){

insInEmpty(data);

}

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

temp->data=data;

temp->next=head;

head=temp;

printf("Inserted successfully.\n");

}

}

void insAtEnd(int data){

if(isEmpty()){

insInEmpty(data);

}

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

struct list \*temp1=(struct list \*)malloc(sizeof(struct list));

temp=head;

while(temp->next!=NULL)

temp=temp->next;

temp1->data=data;

temp1->next=NULL;

temp->next=temp1;

printf("Inserted successfully.\n");

}

}

void insAtPos(int pos,int data){

if(pos==1)

insAtBeg(data);

else{

int c=1;

struct list \*tempData=(struct list \*)malloc(sizeof(struct list));

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

struct list \*prev=(struct list \*)malloc(sizeof(struct list));

tempData->data=data;

temp=head;

while(temp!=NULL){

prev=temp;

temp=temp->next;

c++;

if(c==pos)

break;

}

tempData->next=(temp!=NULL?temp:NULL);

prev->next=tempData;

printf("Inserted successfully.\n");

}

}

void delAtBeg(){

if(isEmpty())

printf("List is empty!!!\n");

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

if(head->next==NULL){

printf("%d Deleted successfully\n",head->data);

free(head);

head=NULL;

}

else{

temp=head;

head=head->next;

printf("%d Deleted successfully\n",temp->data);

free(temp);

temp=NULL;

}

}

}

void delAtEnd(){

if(isEmpty())

printf("List is empty!!!\n");

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

struct list \*prev=(struct list \*)malloc(sizeof(struct list));

if(head->next==NULL){

printf("%d Deleted successfully\n",head->data);

free(head);

head=NULL;

}

else{

temp=head;

while(temp->next!=NULL){

prev=temp;

temp=temp->next;

}

prev->next=NULL;

printf("%d Deleted successfully\n",temp->data);

free(temp);

temp=NULL;

}

}

}

void delAtPos(int pos){

if(pos==1)

delAtBeg();

else{

int c=1;

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

struct list \*prev=(struct list \*)malloc(sizeof(struct list));

temp=head;

while(temp!=NULL && c!=pos){

prev=temp;

temp=temp->next;

c++;

}

if(c==pos){

if(temp->next==NULL)

prev->next=NULL;

else

prev->next=temp->next;

printf("%d Deleted Successfully.\n",temp->data);

free(temp);

temp=NULL;

}

}

}

int count(){

int c=1;

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

temp=head;

while(temp->next!=NULL){

temp=temp->next;

c++;

}

return c;

}

void printList(){

if(isEmpty())

printf("List is empty!!!");

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

temp=head;

while(temp!=NULL){

printf("%d ",temp->data);

temp=temp->next;

}

}

printf("\n");

}

int isEmpty(){

if(head==NULL)

return 1;

else

return 0;

}

void main(){

int choice=1,data,pos;

while(choice!=8){

printf("Enter your choice:\n");

printf("1 : insert at beg\n");

printf("2 : insert at end\n");

printf("3 : insert at pos\n");

printf("4 : delete at beg\n");

printf("5 : delete at end\n");

printf("6 : delete at pos\n");

printf("7 : print list\n");

printf("8 : Exit\n");

scanf("%d",&choice);

clrscr();

switch(choice){

case 1:

printf("Enter data:");

scanf("%d",&data);

insAtBeg(data);

break;

case 2:

printf("Enter data:");

scanf("%d",&data);

insAtEnd(data);

break;

case 3:

printf("Enter Position:");

scanf("%d",&pos);

if(isEmpty() && pos>1)

printf("List is Empty and Invalid Positon!!!\n");

else if(pos<1 || pos>count()+1)

printf("Invalid position!!!\n");

else{

printf("Enter data:");

scanf("%d",&data);

insAtPos(pos,data);

}

break;

case 4:

delAtBeg();

break;

case 5:

delAtEnd();

break;

case 6:

if(isEmpty())

printf("List is Empty!!!\n");

else{

printf("Enter Position:");

scanf("%d",&pos);

if(pos<1 || pos>count())

printf("Invalid position!!!\n");

else

delAtPos(pos);

}

break;

case 7:

printList();

break;

case 8:

exit(0);

break;

default:

printf("Invalid Choice!!!\n");

getch();

exit(0);

}

}

}

Doubly Linked List

#include <stdio.h>

#include <stdlib.h>

struct list{

int data;

struct list \*prev;

struct list \*next;

}\*head;

void insInEmpty(int data){

head = (struct list \*)malloc(sizeof(struct list));

if(head==NULL)

printf("List is Full!!!(overflow)\n");

else{

head->data=data;

head->prev=NULL;

head->next=NULL;

printf("Inserted successfully.\n");

}

}

void insAtBeg(int data){

if(isEmpty()){

insInEmpty(data);

}

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

if(temp==NULL)

printf("List is Full!!!(overflow)\n");

else{

temp->data=data;

temp->prev=NULL;

temp->next=head;

head->prev=temp;

head=temp;

printf("Inserted successfully.\n");

}

}

}

void insAtEnd(int data){

if(isEmpty()){

insInEmpty(data);

}

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

struct list \*tempData=(struct list \*)malloc(sizeof(struct list));

if(temp==NULL || tempData==NULL)

printf("List is Full!!!(overflow)\n");

else{

temp=head;

while(temp->next!=NULL)

temp=temp->next;

tempData->data=data;

tempData->prev=temp;

tempData->next=NULL;

temp->next=tempData;

printf("Inserted successfully.\n");

}

}

}

void insAtPos(int pos,int data){

if(pos==1)

insAtBeg(data);

else{

int c=1;

struct list \*tempData=(struct list \*)malloc(sizeof(struct list));

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

struct list \*prev=(struct list \*)malloc(sizeof(struct list));

if(temp==NULL || tempData==NULL || prev==NULL)

printf("List is Full!!!(overflow)\n");

else{

tempData->data=data;

temp=head;

while(c!=pos){

prev=temp;

temp=temp->next;

c++;

}

tempData->prev=prev;

tempData->next=(temp!=NULL?temp:NULL);

prev->next=tempData;

temp->prev=tempData;

printf("Inserted successfully.\n");

}

}

}

void delAtBeg(){

if(isEmpty())

printf("List is empty!!!\n");

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

if(head->next==NULL){

printf("%d Deleted successfully\n",head->data);

free(head);

head=NULL;

}

else{

temp=head;

head=head->next;

head->prev=NULL;

printf("%d Deleted successfully\n",temp->data);

free(temp);

temp=NULL;

}

}

}

void delAtEnd(){

if(isEmpty())

printf("List is empty!!!\n");

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

struct list \*prev=(struct list \*)malloc(sizeof(struct list));

if(head->next==NULL){

printf("%d Deleted successfully\n",head->data);

free(head);

head=NULL;

}

else{

temp=head;

while(temp->next!=NULL){

prev=temp;

temp=temp->next;

}

prev->next=NULL;

printf("%d Deleted successfully\n",temp->data);

free(temp);

temp=NULL;

}

}

}

void delAtPos(int pos){

if(pos==1)

delAtBeg();

else{

int c=1;

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

struct list \*prev=(struct list \*)malloc(sizeof(struct list));

temp=head;

while(c!=pos){

prev=temp;

temp=temp->next;

c++;

}

if(temp->next==NULL)

prev->next=NULL;

else

prev->next=temp->next;

temp->next->prev=prev;

printf("%d Deleted Successfully.\n",temp->data);

free(temp);

temp=NULL;

}

}

int count(){

int c=1;

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

temp=head;

while(temp->next!=NULL){

temp=temp->next;

c++;

}

return c;

}

void printList(){

if(isEmpty())

printf("List is empty!!!");

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

temp=head;

while(temp!=NULL){

printf("%d ",temp->data);

temp=temp->next;

}

}

printf("\n");

}

int isEmpty(){

if(head==NULL)

return 1;

else

return 0;

}

void main(){

int choice=1,data,pos;

while(choice!=8){

printf("Enter your choice:\n");

printf("1 : insert at beg\n");

printf("2 : insert at end\n");

printf("3 : insert at pos\n");

printf("4 : delete at beg\n");

printf("5 : delete at end\n");

printf("6 : delete at pos\n");

printf("7 : print list\n");

printf("8 : Exit\n");

scanf("%d",&choice);

clrscr();

switch(choice){

case 1:

printf("Enter data:");

scanf("%d",&data);

insAtBeg(data);

break;

case 2:

printf("Enter data:");

scanf("%d",&data);

insAtEnd(data);

break;

case 3:

printf("Enter Position:");

scanf("%d",&pos);

if(isEmpty() && pos>1)

printf("List is Empty and Invalid Positon!!!\n");

else if(pos<1 || pos>count()+1)

printf("Invalid position!!!\n");

else{

printf("Enter data:");

scanf("%d",&data);

insAtPos(pos,data);

}

break;

case 4:

delAtBeg();

break;

case 5:

delAtEnd();

break;

case 6:

if(isEmpty())

printf("List is Empty!!!\n");

else{

printf("Enter Position:");

scanf("%d",&pos);

if(pos<1 || pos>count())

printf("Invalid position!!!\n");

else

delAtPos(pos);

}

break;

case 7:

printList();

break;

case 8:

exit(0);

break;

default:

printf("Invalid Choice!!!\n");

getch();

exit(0);

}

}

}

Circular Linked List

#include <stdio.h>

#include <stdlib.h>

struct list{

int data;

struct list \*next;

}\*head;

void insInEmpty(int data){

head = (struct list \*)malloc(sizeof(struct list));

if(head==NULL)

printf("List is Full!!!(overflow)\n");//only occur when memory is full.

else{

head->data=data;

head->next=head;

printf("Inserted successfully.\n");

}

}

void insAtBeg(int data){

if(isEmpty())

insInEmpty(data);

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

struct list \*tempData=(struct list \*)malloc(sizeof(struct list));

if(tempData==NULL || temp==NULL)

printf("List is Full!!!(overflow)\n");

else{

tempData->data=data;

tempData->next=head;

temp=head;

while(temp->next!=head){

temp=temp->next;

}

temp->next=tempData;

head=tempData;

printf("Inserted successfully.\n");

}

}

}

void insAtEnd(int data){

if(isEmpty())

insInEmpty(data);

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

struct list \*tempData=(struct list \*)malloc(sizeof(struct list));

if(temp==NULL || tempData==NULL)

printf("List is Full!!!(overflow)\n");

else{

tempData->data=data;

tempData->next=head;

temp=head;

while(temp->next!=head)

temp=temp->next;

temp->next=tempData;

printf("Inserted successfully.\n");

}

}

}

void insAtPos(int pos,int data){

if(pos==1)

insAtBeg(data);

else{

int c=1;

struct list \*tempData=(struct list \*)malloc(sizeof(struct list));

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

struct list \*prev=(struct list \*)malloc(sizeof(struct list));

if(tempData==NULL || temp==NULL || prev==NULL)

printf("List is Full!!!(overflow)\n");

else{

tempData->data=data;

temp=head;

do{

prev=temp;

temp=temp->next;

c++;

}while(c!=pos);

tempData->next=(temp->next==head?temp:head);

prev->next=tempData;

printf("Inserted successfully.\n");

}

}

}

void delAtBeg(){

if(isEmpty())

printf("List is Empty!!!\n");

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

struct list \*t=(struct list \*)malloc(sizeof(struct list));

if(head->next==head){

printf("%d Deleted successfully\n",head->data);

free(head);

head=NULL;

}

else{

temp=head;

t=head;

head=head->next;

while(t->next!=temp){

t=t->next;

}

t->next=head;

printf("%d Deleted successfully\n",temp->data);

free(temp);

temp=NULL;

}

}

}

void delAtEnd(){

if(isEmpty())

printf("List is Empty!!!\n");

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

struct list \*prev=(struct list \*)malloc(sizeof(struct list));

if(head->next==head){

printf("%d Deleted successfully\n",head->data);

free(head);

head=NULL;

}

else{

temp=head;

while(temp->next!=head){

prev=temp;

temp=temp->next;

}

prev->next=head;

printf("%d Deleted successfully\n",temp->data);

free(temp);

temp=NULL;

}

}

}

void delAtPos(int pos){

if(pos==1)

delAtBeg();

else{

int c=1;

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

struct list \*prev=(struct list \*)malloc(sizeof(struct list));

temp=head;

do{

prev=temp;

temp=temp->next;

c++;

}while(c!=pos);

if(temp->next==head)

prev->next=head;

else

prev->next=temp->next;

printf("%d Deleted Successfully.\n",temp->data);

free(temp);

temp=NULL;

}

}

int count(){

int c=1;

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

temp=head;

while(temp->next!=head){

c++;

temp=temp->next;

}

return c;

}

void printList(){

if(isEmpty())

printf("List is empty!!!");

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

temp=head;

do{

printf("%d ",temp->data);

temp=temp->next;

}while(temp!=head);

}

printf("\n");

}

int isEmpty(){

if(head==NULL)

return 1;

else

return 0;

}

void main(){

int choice=1,data,pos;

while(choice!=8){

printf("Enter your choice:\n");

printf("1 : insert at beg\n");

printf("2 : insert at end\n");

printf("3 : insert at pos\n");

printf("4 : delete at beg\n");

printf("5 : delete at end\n");

printf("6 : delete at pos\n");

printf("7 : print list\n");

printf("8 : Exit\n");

scanf("%d",&choice);

clrscr();

switch(choice){

case 1:

printf("Enter data:");

scanf("%d",&data);

insAtBeg(data);

break;

case 2:

printf("Enter data:");

scanf("%d",&data);

insAtEnd(data);

break;

case 3:

printf("Enter Position:");

scanf("%d",&pos);

if(isEmpty()&& pos>1)

printf("List is Empty and Invalid Positon!!!\n");

else if(pos<1 || pos>count()+1)

printf("Invalid position!!!\n");

else{

printf("Enter data:");

scanf("%d",&data);

insAtPos(pos,data);

}

break;

case 4:

delAtBeg();

break;

case 5:

delAtEnd();

break;

case 6:

if(isEmpty())

printf("List is Empty!!!\n");

else{

printf("Enter Position:");

scanf("%d",&pos);

if(pos<1 || pos>count())

printf("Invalid position!!!\n");

else

delAtPos(pos);

}

break;

case 7:

printList();

break;

case 8:

exit(0);

break;

default:

printf("Invalid Choice!!!\n");

getch();

exit(0);

}

}

}

Stack Array

#include <stdio.h>

#define MAX 15

int top=-1;

int stack[MAX];

int isEmpty(){

return top == -1;

}

int isFull(){

return top == MAX-1;

}

int peek(){

if(!isEmpty()){

printf("Peek Element is : %d.\n",stack[top]);

return stack[top];

}

else{

printf("Stack is Empty!!!\n");

return -1;

}

}

void push(){

if(!isFull()){

int data;

printf("Enter data:");

scanf("%d",&data);

stack[++top]=data;

printf("Inserted Successfully.\n");

}

else

printf("Stack is Full(overflow)!!!\n");

}

int pop(){

if(!isEmpty()){

printf("%d Deleted Successfully.\n",stack[top]);

return stack[top--];

}

else{

printf("Stack is Empty(underflow)!!!\n");

return -1;

}

}

void printStack(){

if(!isEmpty()){

int temp=top;

for(;temp>=0;temp--){

printf("%d ",stack[temp]);

}

}

else

printf("Stack is Empty!!!");

printf("\n");

}

void main(){

int data;

int choice=0;

while(choice!=8){

printf("Enter your choice:\n");

printf("1 : Check stack it is empty or not.\n");

printf("2 : Check stack it is full or not.\n");

printf("3 : Peek Element.\n");

printf("4 : Push Operation.\n");

printf("5 : Pop Operation.\n");

printf("6 : Print Stack.\n");

printf("7 : Exit.\n");

scanf("%d",&choice);

clrscr();

switch(choice){

case 1:

if(isEmpty())

printf("Stack is Empty!!!\n");

else

printf("Stack is not Empty!!!\n");

break;

case 2:

if(isFull())

printf("Stack is Full!!!\n");

else

printf("Stack is not Full!!!\n");

break;

case 3:

peek();

break;

case 4:

push();

break;

case 5:

pop();

break;

case 6:

printStack();

break;

case 7:

exit(0);

break;

default:

printf("Invalid Choice!!!\n");

getch();

exit(0);

}

}

}

Stack LinkedList

#include <stdio.h>

#include <stdlib.h>

struct stack{

int data;

struct stack \*prev;

}\*top;

int isEmpty(){

return top == NULL;

}

int peek(){

if(!isEmpty()){

printf("Peek Element is : %d.\n",top->data);

return top->data;

}

else{

printf("Stack is Empty!!!\n");

return -1;

}

}

void push(){

struct stack \*temp=(struct stack \*)malloc(sizeof(struct stack));

if(temp==NULL)

printf("Stack is Full(overflow)!!!\n");

else{

int data;

printf("Enter data:");

scanf("%d",&data);

temp->data=data;

if(top==NULL)

temp->prev=NULL;

else

temp->prev=top;

top=temp;

printf("Inserted Successfully.\n");

}

}

int pop(){

if(!isEmpty()){

int t=top->data;

struct stack \*temp=(struct stack \*)malloc(sizeof(struct stack));

printf("%d Deleted Successfully.\n",top->data);

temp=top;

if(top->prev==NULL)

top=NULL;

else

top=top->prev;

free(temp);

temp=NULL;

return t;

}

else{

printf("Stack is Empty(underflow)!!!\n");

return -1;

}

}

void printStack(){

struct stack \*temp=(struct stack \*)malloc(sizeof(struct stack));

if(!isEmpty()){

temp=top;

while(temp!=NULL){

printf("%d ",temp->data);

temp=temp->prev;

}

}

else

printf("Stack is Empty!!!");

printf("\n");

}

void main(){

int choice=0;

while(choice!=8){

printf("Enter your choice:\n");

printf("1 : Check stack it is empty or not.\n");

printf("2 : Peek Element.\n");

printf("3 : Push Operation.\n");

printf("4 : Pop Operation.\n");

printf("5 : Print Stack.\n");

printf("6 : Exit.\n");

scanf("%d",&choice);

clrscr();

switch(choice){

case 1:

if(isEmpty())

printf("Stack is Empty!!!\n");

else

printf("Stack is not Empty!!!\n");

break;

case 2:

peek();

break;

case 3:

push();

break;

case 4:

pop();

break;

case 5:

printStack();

break;

case 6:

exit(0);

break;

default:

printf("Invalid Choice!!!\n");

getch();

exit(0);

}

}

}

Queue Array

#include <stdio.h>

#define MAX 15

int front=-1;

int rear=-1;

int queue[MAX];

int isEmpty(){

return front == -1;

}

int isFull(){

return rear==(MAX-1);

}

int peek(){

if(!isEmpty()){

printf("Peek Element is : %d.\n",queue[front]);

return queue[front];

}

else{

printf("Queue is Empty!!!\n");

return -1;

}

}

void enqueue(){

if(!isFull()){

int data;

printf("Enter data:");

scanf("%d",&data);

if(isEmpty()){

front++;

queue[++rear]=data;

}

else{

queue[++rear]=data;

}

printf("Inserted Successfully.\n");

}

else

printf("Queue is Full!!!\n");

}

int dequeue(){

if(!isEmpty()){

printf("%d Deleted Successfully.\n",queue[front]);

if(front==rear){

int temp=front;

front=rear=-1;

return queue[temp];

}

else

return queue[front++];

}

else{

printf("Queue is Empty!!!\n");

return -1;

}

}

void printQueue(){

if(!isEmpty()){

int temp=front;

for(;temp<=rear;temp++){

printf("%d ",queue[temp]);

}

}

else

printf("Queue is Empty!!!");

printf("\n");

}

void main(){

int data;

int choice=0;

while(choice!=8){

printf("Enter your choice:\n");

printf("1 : Check Queue it is empty or not.\n");

printf("2 : Check Queue it is full or not.\n");

printf("3 : Peek Element.\n");

printf("4 : Enqueue Operation.\n");

printf("5 : Dequeue Operation.\n");

printf("6 : Print Queue.\n");

printf("7 : Exit.\n");

scanf("%d",&choice);

clrscr();

switch(choice){

case 1:

if(isEmpty())

printf("Queue is Empty!!!\n");

else

printf("Queue is not Empty!!!\n");

break;

case 2:

if(isFull())

printf("Queue is Full!!!\n");

else

printf("Queue is not Full!!!\n");

break;

case 3:

peek();

break;

case 4:

enqueue();

break;

case 5:

dequeue();

break;

case 6:

printQueue();

break;

case 7:

exit(0);

break;

default:

printf("Invalid Choice!!!\n");

getch();

exit(0);

}

}

}

Queue Linked List

#include <stdio.h>

struct queue{

int data;

struct queue \*next;

}\*front,\*rear;

int isEmpty(){

return front == NULL;

}

int peek(){

if(!isEmpty()){

printf("Peek Element is : %d.\n",front->data);

return front->data;

}

else{

printf("Queue is Empty!!!\n");

return -1;

}

}

void enqueue(){

struct queue \*temp=(struct queue \*)malloc(sizeof(struct queue));

if(temp==NULL)

printf("Queue is Full(overflow)!!!\n");

else{

int data;

printf("Enter data:");

scanf("%d",&data);

temp->data=data;

temp->next=NULL;

if(isEmpty())

front=rear=temp;

else

rear->next=temp;

rear=temp;

printf("Inserted Successfully.\n");

}

}

int dequeue(){

if(!isEmpty()){

int t=front->data;

struct queue \*temp=(struct queue \*)malloc(sizeof(struct queue));

printf("%d Deleted Successfully.\n",front->data);

temp=front;

if(front->next==NULL)

front=NULL;

else

front=front->next;

free(temp);

temp=NULL;

return t;

}

else{

printf("Queue is Empty(underflow)!!!\n");

return -1;

}

}

void printQueue(){

struct queue \*temp=(struct queue \*)malloc(sizeof(struct queue));

if(!isEmpty()){

temp=front;

while(temp!=NULL){

printf("%d ",temp->data);

temp=temp->next;

}

}

else

printf("Queue is Empty!!!");

printf("\n");

}

void main(){

int data;

int choice=0;

while(choice!=8){

printf("Enter your choice:\n");

printf("1 : Check Queue it is empty or not.\n");

printf("2 : Peek Element.\n");

printf("3 : Enqueue Operation.\n");

printf("4 : Dequeue Operation.\n");

printf("5 : Print Queue.\n");

printf("6 : Exit.\n");

scanf("%d",&choice);

clrscr();

switch(choice){

case 1:

if(isEmpty())

printf("Queue is Empty!!!\n");

else

printf("Queue is not Empty!!!\n");

break;

case 2:

peek();

break;

case 3:

enqueue();

break;

case 4:

dequeue();

break;

case 5:

printQueue();

break;

case 6:

exit(0);

break;

default:

printf("Invalid Choice!!!\n");

getch();

exit(0);

}

}

}

Circular Queue Array

#include <stdio.h>

#define MAX 15

int front=-1;

int rear=-1;

int queue[MAX];

int isEmpty(){

if(front==-1){

printf("Queue is Empty!!!\n");

return 1;

}

else

return 0;

}

int isFull(){

if((rear+1)%MAX==front){

printf("Queue is Full!!!\n");

return 1;

}

else

return 0;

}

int peek(){

if(!isEmpty()){

printf("Peek Element is : %d.\n",queue[front]);

return queue[front];

}

else

return -1;

}

void enqueue(){

if(!isFull()){

int data;

printf("Enter data:");

scanf("%d",&data);

if(isEmpty()){

front++;

queue[++rear]=data;

}

else{

rear=(rear+1)%MAX;

queue[rear]=data;

}

printf("Inserted Successfully.\n");

}

}

int dequeue(){

if(!isEmpty()){

int temp=front;

printf("%d Deleted Successfully.\n",queue[front]);

if(front==rear){

front=rear=-1;

return queue[temp];

}

else{

front=(front+1)%MAX;

return queue[temp];

}

}

else

return -1;

}

void printQueue(){

if(!isEmpty()){

int temp=front;

while(temp!=rear){

printf("%d ",queue[temp]);

temp=(temp+1)%MAX;

}

printf("%d ",queue[temp]);

}

printf("\n");

}

void main(){

int data;

int choice=0;

while(choice!=8){

printf("Enter your choice:\n");

printf("1 : Check Queue it is empty or not.\n");

printf("2 : Check Queue it is full or not.\n");

printf("3 : Peek Element.\n");

printf("4 : Enqueue Operation.\n");

printf("5 : Dequeue Operation.\n");

printf("6 : Print Queue.\n");

printf("7 : Exit.\n");

scanf("%d",&choice);

clrscr();

switch(choice){

case 1:

if(!isEmpty())

printf("Queue is not Empty!!!\n");

break;

case 2:

if(!isFull())

printf("Queue is not Full!!!\n");

break;

case 3:

peek();

break;

case 4:

enqueue();

break;

case 5:

dequeue();

break;

case 6:

printQueue();

break;

case 7:

exit(0);

break;

default:

printf("Invalid Choice!!!\n");

getch();

exit(0);

}

}

}

Circular Queue LinkedList

#include <stdio.h>

#include <stdlib.h>

struct queue{

int data;

struct queue \*next;

}\*front,\*rear;

int isEmpty(){

if(front==NULL){

printf("Queue is Empty!!!\n");

return 1;

}

else

return 0;

}

int peek(){

if(!isEmpty()){

printf("Peek Element is : %d.\n",front->data);

return front->data;

}

else

return -1;

}

void enqueue(){

struct queue \*temp=(struct queue \*)malloc(sizeof(struct queue));

if(temp==NULL)

printf("Queue is Full(overflow)!!!\n");

else{

int data;

printf("Enter data:");

scanf("%d",&data);

temp->data=data;

if(rear==NULL){

temp->next=temp;

front=temp;

}

else{

temp->next=front;

rear->next=temp;

}

rear=temp;

printf("Inserted Successfully.\n");

}

}

int dequeue(){

if(!isEmpty()){

int t=front->data;

struct queue \*temp=(struct queue \*)malloc(sizeof(struct queue));

printf("%d Deleted Successfully.\n",t);

temp=front;

if(front->next==front)

front=rear=NULL;

else{

front=front->next;

rear->next=front;

}

free(temp);

temp=NULL;

return t;

}

else

return -1;

}

void printQueue(){

struct queue \*temp=(struct queue \*)malloc(sizeof(struct queue));

if(!isEmpty()){

temp=front;

do{

printf("%d ",temp->data);

temp=temp->next;

}while(temp!=front);

}

else

printf("Queue is Empty!!!");

printf("\n");

}

void main(){

int data;

int choice=0;

while(choice!=8){

printf("Enter your choice:\n");

printf("1 : Check Queue it is empty or not.\n");

printf("2 : Peek Element.\n");

printf("3 : Enqueue Operation.\n");

printf("4 : Dequeue Operation.\n");

printf("5 : Print Queue.\n");

printf("6 : Exit.\n");

scanf("%d",&choice);

clrscr();

switch(choice){

case 1:

if(!isEmpty())

printf("Queue is not Empty!!!\n");

break;

case 2:

peek();

break;

case 3:

enqueue();

break;

case 4:

dequeue();

break;

case 5:

printQueue();

break;

case 6:

exit(0);

break;

default:

printf("Invalid Choice!!!\n");

getch();

exit(0);

}

}

}

Dequeue Array Input Restriction

#include <stdio.h>

#define MAX 10

//Input Restricted dequeue by darshil solanki

int front=-1;

int rear=-1;

int queue[MAX];

int isEmpty(){

return front == -1;

}

int isFull(){

return (rear+1)%MAX==front;

}

int peek(){

if(!isEmpty()){

printf("Peek Element is : %d.\n",queue[front]);

return queue[front];

}

else{

printf("Queue is Empty!!!\n");

return -1;

}

}

void enqueue(){

if(!isFull()){

int data;

printf("Enter data:");

scanf("%d",&data);

if(isEmpty()){

front++;

queue[++rear]=data;

}

else{

rear=(rear+1)%MAX;

queue[rear]=data;

}

printf("Inserted Successfully.\n");

}

else

printf("Queue is Full(overflow)!!!\n");

}

int dequeueAtBeg(){

if(!isEmpty()){

int temp=front;

printf("%d Deleted Successfully.\n",queue[front]);

if(front==rear){

front=rear=-1;

return queue[temp];

}

else

front=(front+1)%MAX;

return queue[temp];

}

else{

printf("Queue is Empty(underflow)!!!\n");

return -1;

}

}

int dequeueAtEnd(){

if(!isEmpty()){

int temp=rear;

printf("%d Deleted Successfully.\n",queue[rear]);

if(front==rear){

front=rear=-1;

return queue[temp];

}

else{

rear=(rear-1)%MAX!=-1?rear-1:MAX-1;

return queue[temp];

}

}

else{

printf("Queue is Empty(underflow)!!!\n");

return -1;

}

}

void printQueue(){

if(!isEmpty()){

int temp=front;

while(temp!=rear){

printf("%d ",queue[temp]);

temp=(temp+1)%MAX;

}

printf("%d ",queue[temp]);

}

else

printf("Queue is Empty!!!\n");

printf("\n");

}

void main(){

int data;

int choice=0;

while(choice!=8){

printf("Enter your choice:\n");

printf("1 : Check Queue it is empty or not.\n");

printf("2 : Check Queue it is full or not.\n");

printf("3 : Peek Element.\n");

printf("4 : Enqueue Operation.\n");

printf("5 : Dequeue from front Operation.\n");

printf("6 : Dequeue from rear Operation.\n");

printf("7 : Print Queue.\n");

printf("8 : Exit.\n");

scanf("%d",&choice);

clrscr();

switch(choice){

case 1:

if(isEmpty())

printf("Queue is Empty!!!\n");

else

printf("Queue is not Empty!!!\n");

break;

case 2:

if(isFull())

printf("Queue is Full!!!\n");

else

printf("Queue is not Full!!!\n");

break;

case 3:

peek();

break;

case 4:

enqueue();

break;

case 5:

dequeueAtBeg();

break;

case 6:

dequeueAtEnd();

break;

case 7:

printQueue();

break;

case 8:

exit(0);

break;

default:

printf("Invalid Choice!!!\n");

getch();

exit(0);

}

}

}

Dequeue Array Output Restriction

#include <stdio.h>

#define MAX 10

//Output Restricted dequeue by darshil solanki

int front=-1;

int rear=-1;

int queue[MAX];

int isEmpty(){

return front == -1;

}

int isFull(){

return (rear+1)%MAX==front;

}

int peek(){

if(!isEmpty()){

printf("Peek Element is : %d.\n",queue[front]);

return queue[front];

}

else{

printf("Queue is Empty!!!\n");

return -1;

}

}

int dequeue(){

if(!isEmpty()){

int temp=front;

printf("%d Deleted Successfully.\n",queue[front]);

if(front==rear){

front=rear=-1;

return queue[temp];

}

else{

front=(front+1)%MAX;

return queue[temp];

}

}

else{

printf("Queue is Empty(underflow)!!!\n");

return -1;

}

}

void enqueueAtBeg(){

if(!isFull()){

int data;

printf("Enter data:");

scanf("%d",&data);

if(isEmpty()){

rear++;

queue[++front]=data;

}

else{

front=(front-1)%MAX!=-1?front-1:MAX-1;

queue[front]=data;

}

printf("Inserted Successfully.\n");

}

else

printf("Queue is Full(overflow)!!!\n");

}

void enqueueAtEnd(){

if(!isFull()){

int data;

printf("Enter data:");

scanf("%d",&data);

if(isEmpty()){

front++;

queue[++rear]=data;

}

else{

rear=(rear+1)%MAX;

queue[rear]=data;

}

printf("Inserted Successfully.\n");

}

else

printf("Queue is Full(overflow)!!!\n");

}

void printQueue(){

if(!isEmpty()){

int temp=front;

while(temp!=rear){

printf("%d ",queue[temp]);

temp=(temp+1)%MAX;

}

printf("%d ",queue[temp]);

}

else

printf("Queue is Empty!!!\n");

printf("\n");

}

void main(){

int data;

int choice=0;

while(choice!=8){

printf("Enter your choice:\n");

printf("1 : Check Queue it is empty or not.\n");

printf("2 : Check Queue it is full or not.\n");

printf("3 : Peek Element.\n");

printf("4 : Enqueue from front Operation.\n");

printf("5 : Enqueue from rear Operation.\n");

printf("6 : Dequeue Operation.\n");

printf("7 : Print Queue.\n");

printf("8 : Exit.\n");

scanf("%d",&choice);

clrscr();

switch(choice){

case 1:

if(isEmpty())

printf("Queue is Empty!!!\n");

else

printf("Queue is not Empty!!!\n");

break;

case 2:

if(isFull())

printf("Queue is Full!!!\n");

else

printf("Queue is not Full!!!\n");

break;

case 3:

peek();

break;

case 4:

enqueueAtBeg();

break;

case 5:

enqueueAtEnd();

break;

case 6:

dequeue();

break;

case 7:

printQueue();

break;

case 8:

exit(0);

break;

default:

printf("Invalid Choice!!!\n");

getch();

exit(0);

}

}

}

Dequeue LinkedList Input Restriction

#include <stdio.h>

#include <stdlib.h>

struct queue{

int data;

struct queue \*next;

}\*front,\*rear;

int isEmpty(){

return rear==NULL;

}

int peek(){

if(!isEmpty()){

printf("Peek Element is : %d.\n",front->data);

return front->data;

}

else{

printf("Queue is Empty!!!\n");

return -1;

}

}

void enqueue(){

struct queue \*tempData=(struct queue \*)malloc(sizeof(struct queue));

if(tempData==NULL)

printf("Dequeue is Full!!!(overflow)\n");

else{

int data;

printf("Enter data:");

scanf("%d",&data);

tempData->data=data;

tempData->next=NULL;

if(isEmpty())

front=tempData;

else

rear->next=tempData;

rear=tempData;

printf("Inserted Successfully.\n");

}

}

int dequeueAtBeg(){

if(isEmpty()){

printf("Queue is empty!!!\n");

return -1;

}

else{

int t=front->data;

struct queue \*temp=(struct queue \*)malloc(sizeof(struct queue));

temp=front;

if(temp->next==NULL){

printf("%d Deleted successfully\n",front->data);

front=rear=NULL;

}

else{

front=front->next;

printf("%d Deleted successfully\n",temp->data);

}

free(temp);

temp=NULL;

return t;

}

}

int dequeueAtEnd(){

if(isEmpty()){

printf("Queue is empty!!!\n");

return -1;

}

else{

int t=rear->data;

struct queue \*temp=(struct queue \*)malloc(sizeof(struct queue));

struct queue \*prev=(struct queue \*)malloc(sizeof(struct queue));

temp=front;

if(temp->next==NULL){

printf("%d Deleted successfully\n",front->data);

rear=NULL;

}

else{

while(temp->next!=NULL){

prev=temp;

temp=temp->next;

}

prev->next=NULL;

printf("%d Deleted successfully\n",temp->data);

}

free(temp);

temp=NULL;

return t;

}

}

void printQueue(){

if(isEmpty())

printf("Queue is empty!!!");

else{

struct queue \*temp=(struct queue \*)malloc(sizeof(struct queue));

temp=front;

while(temp!=NULL){

printf("%d ",temp->data);

temp=temp->next;

};

}

printf("\n");

}

void main(){

int choice=1,data,pos;

while(choice!=8){

printf("Enter your choice:\n");

printf("1 : Check Queue it is empty or not.\n");

printf("2 : Peek Element.\n");

printf("3 : Enqueue Operation.\n");

printf("4 : Dequeue at beginning Operation.\n");

printf("5 : Dequeue at ending Operation.\n");

printf("6 : Print Queue.\n");

printf("7 : Exit.\n");

scanf("%d",&choice);

clrscr();

switch(choice){

case 1:

if(isEmpty())

printf("Queue is Empty!!!\n");

else

printf("Queue is not Empty!!!\n");

break;

case 2:

peek();

break;

case 3:

enqueue();

break;

case 4:

dequeueAtBeg();

break;

case 5:

dequeueAtEnd();

break;

case 6:

printQueue();

break;

case 7:

exit(0);

break;

default:

printf("Invalid Choice!!!\n");

getch();

exit(0);

}

}

}

Dequeue LinkedList Output Restriction

#include <stdio.h>

#include <stdlib.h>

struct queue{

int data;

struct queue \*next;

}\*front,\*rear;

int isEmpty(){

return front == NULL;

}

int peek(){

if(!isEmpty()){

printf("Peek Element is : %d.\n",front->data);

return front->data;

}

else{

printf("Queue is Empty!!!\n");

return -1;

}

}

int dequeue(){

if(!isEmpty()){

int t=front->data;

struct queue \*temp=(struct queue \*)malloc(sizeof(struct queue));

temp=front;

printf("%d Deleted successfully\n",front->data);

if(temp->next==NULL)

front=rear=NULL;

else

front=front->next;

free(temp);

temp=NULL;

return t;

}

else{

printf("Queue is Empty!!!(underflow)\n");

return -1;

}

}

void enqueueAtBeg(){

struct queue \*tempData=(struct queue \*)malloc(sizeof(struct queue));

if(tempData==NULL)

printf("Dequeue is Full!!!(overflow)\n");

else{

int data;

printf("Enter data:");

scanf("%d",&data);

tempData->data=data;

tempData->next=NULL;

if(isEmpty())

rear=tempData;

else

tempData->next=front;

front=tempData;

printf("Inserted Successfully.\n");

}

}

void enqueueAtEnd(){

struct queue \*tempData=(struct queue \*)malloc(sizeof(struct queue));

if(tempData==NULL)

printf("Dequeue is Full!!!(overflow)\n");

else{

int data;

printf("Enter data:");

scanf("%d",&data);

tempData->data=data;

tempData->next=NULL;

if(isEmpty())

front=tempData;

else

rear->next=tempData;

rear=tempData;

printf("Inserted Successfully.\n");

}

}

void printQueue(){

if(isEmpty())

printf("Queue is empty!!!");

else{

struct queue \*temp=(struct queue \*)malloc(sizeof(struct queue));

temp=front;

while(temp!=NULL){

printf("%d ",temp->data);

temp=temp->next;

};

}

printf("\n");

}

void main(){

int data;

int choice=0;

while(choice!=8){

printf("Enter your choice:\n");

printf("1 : Check Queue it is empty or not.\n");

printf("2 : Peek Element.\n");

printf("3 : Enqueue at beginning Operation.\n");

printf("4 : Enqueue at ending Operation.\n");

printf("5 : Dequeue Operation.\n");

printf("6 : Print Queue.\n");

printf("7 : Exit.\n");

scanf("%d",&choice);

clrscr();

switch(choice){

case 1:

if(isEmpty())

printf("Queue is Empty!!!\n");

else

printf("Queue is not Empty!!!\n");

break;

case 2:

peek();

break;

case 3:

enqueueAtBeg();

break;

case 4:

enqueueAtEnd();

break;

case 5:

dequeue();

break;

case 6:

printQueue();

break;

case 7:

exit(0);

break;

default:

printf("Invalid Choice!!!\n");

getch();

exit(0);

}

}

}